# RETROSPECTIVE: Memory Persistency

Stephen Pelley*, Peter M. Chen†, Thomas F. Wenisch‡,
*Unaffiliated †U. Michigan ‡Google

## I. Summary

Byte-addressable nonvolatile memories can enable high performance recoverable systems. These technologies pair the high performance of DRAM with the durability of disk and Flash memory, providing systems with memory performance approaching that of DRAM, yet recoverability after failure. However, ensuring proper recovery requires constraints on the ordering of writes. In 2014, DRAM interconnects lacked the interface to describe and enforce write ordering constraints; ordering constraints that arise from memory consistency requirements are usually enforced at the processor, which is insufficient for failure tolerance with acceptable performance.

In our 2014 ISCA paper, we introduced Memory Persistency, a framework motivated by memory consistency to provide an interface for enforcing the order writes become durable, an operation we refer to as a "persist." Just as memory consistency constrains the visible order of loads and stores between processors, so memory persistency constrains the order of persists with respect to one another and to loads and stores. These constraints allow the programmer to reason about the ordering of persists thereby guarantee correct recovery from system failures. Both memory consistency and memory persistency models define an interface and set of memory order guarantees for the programmer while permitting different implementations and optimizations. Both types of models trade off increased concurrency and performance for increased burden on the programmer to insert correct annotations.

Our work introduced the abstraction of a *recovery observer* that atomically reads the entire persistent memory address space at the moment of failure. Ordering constraints for correct recovery thus become ordering constraints on memory and persist operations as viewed from the recovery observer. With this abstraction, we can apply the reasoning tools of memory consistency to persistency—any two stores to the persistent memory address space that are ordered with respect to the recovery observer imply an ordering constraint on the corresponding persists. Conversely, stores that are not ordered with respect to the observer allow corresponding persists to be reordered or performed in parallel.

In our ISCA 2014 paper, we define memory persistency, describe the design space of memory persistency models, and evaluate several persistency models. In particular, we introduce the notion of *strict* persistency, where persistent memory order is identical to volatile memory order, and *relaxed* persistency, where these two orders are allowed to diverge. Relaxing persistency allows systems with conservative consistency, such as sequential consistency, to improve persist concurrency. Layering relaxed persistency on strict consistency allows programmers to write synchronization code with the more intuitive interface of strict consistency while still expressing high concurrency for the much slower persist operations.

We describe memory persistency and follow-on research in greater detail in *A Primer on Memory Persistency* [3].

## II. Historical context

Starting with the seminal work on sequential consistency by Lamport in 1979 [6], a long sequence of computer architecture and programming language research formalized the notion of memory consistency models, to allow programmers to reason about the correctness of multiprocessor programs in light of the increasingly bewildering behaviors of processor architectures with out-of-order execution and complex memory hierarchies, where loads and stores did not necessarily propagate throughout the memory system in the order that a program specified. Interestingly, hardware design, introducing new mechanisms like speculative loads and post-retirement store buffers, often preceded development of the theory of how to correctly synchronize multiprocessor programs under these mechanisms. For example, a sound specification of the Intel x86 memory model [9] was not published until decades after x86-based multiprocessors were commercially sold.

Similarly, clever device designers conceived ways to make memory devices and arrays that, like DRAM, support a byte-addressable load-store architecture, but retain their state when supply voltage is removed. The first publication to examine how such devices might be incorporated into computer architectures was, to the best of our knowledge, the work of Benjamin Lee and Microsoft co-authors at ISCA 2009 [7]. Lee surveyed a series of articles from the devices community on phase change memory and evaluated what performance characteristics might be achieved if these devices supplanted DRAM-based main memory. They were bullish on the new capabilities a persistent main memory might offer. They observed, "Software cognizant of this newly provided persistance can provide qualitatively new capabilities. For example, system boot/hibernate will be perceived as instantaneous; application checkpointing will be inexpensive; file systems will provide stronger safety guarantees."

Lee's ISCA paper focused on a variety of technical challenges with phase change memory (PCM) technology that needed to be addressed to deploy PCM-based memory devices, but did not consider the system-level challenges that must be addressed to make use of PCM's durability. An overlapping team of authors proposed to build a file system using byte-addressable persistent memory at SOSP the same year [2].

That work observed that file system correctness depends upon the ordering of write operations, and yet existing processor architectures lacked any mechanism to allow programmer control of when data is evicted from caches and sent to memory, short of marking an address region uncacheable. They proposed to introducing a new abstraction called an epoch barrier to divide programmer order into regions where the persistent writes within a region are mutually unordered, but all writes in one region must persist before any in a subsequent region. This new epoch barrier is analagous to fence instructions in many memory consistency models and, in the terminology of our later work, the resulting persistency model is called epoch persistency.

In reading these early works on persistent memory, our team recognized that epoch persistency is only one of many possible programming models for enforcing correct event order for byte-addressable persistent memory. We recognized that a more comprehensive theory might unlock additional design points in the trade-off between simplicity of programming and concurrency among slow, expensive persist operations. Moreover, our team had spent many years studying speculative implementations of memory consistency (e.g., Invisifence [1]). The key insight of this line of research is that the ordering rules of memory consistency models need only be obeyed if there is a racing memory access from another processor that can observe the misordering. In the immortal words of Bart Simpson, "I didn't do it. Nobody saw me do it. There's no way you can prove anything!" [4]. This insight lead us to conceive the *recovery observer* as the basis to apply the collective techniques of the memory consistency literature to this new problem domain. We coined the term "memory persistency" to suggest the similarities to memory consistency, but also to emphasize that memory persistency gives rise to a new event ordering that is distinct from visibility ordering governed by consistency.

## III. 9 YEARS LATER

A variety of follow-on research builds upon our theory of memory persistency, proposing new models and implementations. We describe these in greater detail in our *Primer* [3].

In seeking to commercialize byte-addressable persistent memory, Intel, too, recognized the need for architectural extensions to describe persist ordering. Intel introduced the *pcommit* instruction to enable this control. However, we and others recognized that persistency model implied by pcommit leads to inherently poor performance [5]. The instruction has since been abandoned in favor of much simpler mechanisms that guarantee in-flight writes will be flushed upon power failure [8].

The most significant attempt to commercialize persistent memory since these early publications is Intel's OPTANE product line, which introduced persistent memory in a form-factor compatible with existing DRAM memory systems. However, this product line was cancelled in July of 2022, with a write-down of $559 million in unsold inventory. Intel's statements at the time indicate an anticipated shift to compute express link (CXL)-based attachment of new memory technology in future systems rather than a drop-in replacement for DRAM. We look forward to this reintroduction of persistent memory; examining the system-level challenges of optimizing for this interface is a ripe area for additional research.

While we have limited visibility into why this particular product line was not a commercial success, we can nonetheless offer some observations on why it is challenging to introduce a new persistent storage tier, especially in the current context of global hyperscale cloud computing. To be considered safely durable, it is not enough for data to be stored on media that preserves state across power failures, data must also be replicated to multiple distinct failure domains (i.e., different systems/disks). Ideally, important data is replicated to geographically diverse data centers to prevent data loss even in the case of disasters that bring an entire site offline. Once one must incur the cost of replicating data over the network, the latency and byte-addressability properties of persistent memory matter less when weighed against the cost advantages of cheaper storage media, like NAND Flash. We nevertheless hope that continued research will uncover new opportunities for these memory technologies in the consumer/portable device space or in as yet unexplored system designs in the cloud.

## REFERENCES

[1] C. Blundell, M. M. Martin, and T. F. Wenisch, "Invisifence: performance-transparent memory ordering in conventional multiprocessors," in *Proceedings of the 36th annual International Symposium on Computer architecture*, 2009, pp. 233–244.

[2] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, "Better i/o through byte-addressable, persistent memory," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 133–146.

[3] V. Gogte, A. Kolli, and T. F. Wenisch, "A primer on memory persistency," *Synthesis Lectures on Computer Architecture*, vol. 17, no. 1, 2022.

[4] M. Groening, J. L. Brooks, and S. Simon, "Bart gets famous," *The Simpsons*, vol. 5, no. 12.

[5] A. Kolli, J. Rosen, S. Diestelhorst, A. Saidi, S. Pelley, S. Liu, P. M. Chen, and T. F. Wenisch, "Delegated persist ordering," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.

[6] Lamport, "How to make a multiprocessor computer that correctly executes multiprocess programs," *IEEE transactions on computers*, vol. 100, no. 9, pp. 690–691, 1979.

[7] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *Proceedings of the 36th annual international symposium on Computer architecture*, 2009, pp. 2–13.

[8] A. Rudoff, "Deprecating the PCOMMIT instruction." [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/deprecate-pcommit-instruction.html

[9] P. Sewell, S. Sarkar, S. Owens, F. Z. Nardelli, and M. O. Myreen, "x86-tso: a rigorous and usable programmer's model for x86 multiprocessors," *Communications of the ACM*, vol. 53, no. 7, pp. 89–97, 2010.